Санкт-Петербургский государственный университет

Кафедра Моделирования Экономических Систем

Пучнин Николай Алексеевич

Выпускная квалификационная работа бакалавра

Определение дефектов пиломатериалов методом компьютерного зрения

Направление 02.03.02

Фундаментальная информатика и информационные технологии

Научный руководитель:
кандидат ф.-м. н., доцент Ковшов А. М.

Санкт-Петербург
2017

SAINT-PETERSBURG STATE UNIVERSITY

Fundamental informatics and information technologies
Programing and information technologies

Nikolay Puchnin

Identification of timber defects by computer vision

Graduation qualification work

Scientific supervisor:
candidate of Physico-Mathematical Sciences,
Associate Professor A.M. Kovshov

Saint-Petersburg
2017

Оглавление

Введение

Компьютерное зрение - это набор методов и технологий, позволяющий компьютерам не просто обрабатывать изображения как массив данных, а воспринимать их и интерпретировать подобным человеку образом. Все популярнее оно становится в промышленности, так как при помощи подобных методов можно автоматизировать и существенно улучшить процесс, за которым нужен визуальный контроль. Такими процессами могут выступать контроль качества продукта (например, контроль качества железобетонных конструкций, нефтепродуктов, зерна и др.), всевозможные измерения, подсчет количества объектов и многое другое. Особенно важно, что в этой сфере человеческой деятельности уже дано строгое определение задачам, решаемым методами компьютерного зрения, и с течением времени количество этих задач только увеличивается.

В рамках данной работы мы рассматриваем одну из подобных задач, а именно определение дефектов на пиломатериалах, и приводим ей сопутствующее решение при помощи компьютерных технологий. Почти на каждом деревообрабатывающем предприятии осуществляется сортировка с учетом дефектов на полотне доски. Но эта сортировка по качеству все еще часто происходит вручную, когда работник самостоятельно оценивает каждую из досок. Поэтому нельзя сбрасывать со счетов человеческий фактор, потому что на результаты влияет опыт работника, его навыки и многое другое, кроме того, он не может превысить пределы своих возможностей. В итоге, скорость сортировки упирается в быстроту оценки качества человеком, что серьезно задерживает весь процесс. Но скорость и качество процесса можно серьезно повысить, если использовать для решения этой задачи методы компьютерного зрения и машинного обучения. Таким образом, исключается необходимость работникам заниматься монотонной работой и они могут переключить свои силы на решение таких задач производства, какие в силах решить только человек.

1. Постановка задачи

Как это уже было сказано в введении, перед нами стоит задача создать систему определения пороков древесины по их фотографиям. Для реализации такой системы нам будет необходим компьютер, фотоаппарат или видеокамера. Зная скорость перемещения пиломатериалов по конвееру, легко определить необходимый интервал времени, через который происходит фотографирование полотна таким образом, что рассматриваются все части досок. Если же мы используем видеокамеру, то задача все равно сводится к анализу фотографий, ведь каждый отдельно взятый кадр является картинкой. Поэтому, взяв из видеопотока кадры с каким-то промежутком, чтобы рассматривать только новые части полотна доски, а не несколько десятков кадров одного и того же места, мы решаем ту же самую задачу.

а) Пример доски

б) Пример дефекта

Рис. 1: Пример доски и дефекта

Основная задача состоит в локализации и классификации дефектов с помощью выбранных алгоритмов компьютерного зрения и машинного обучения на изображении, полученном на вход. Для решения этой задачи уже существуют данные, хранящие в себе файлы изображений досок, типы дефектов на каждом изображении и местоположения этих дефектов в виде массива пар точек (x, y) в прямоугольной системе координат. Всего дефектов насчитывается одиннадцать классов, которые представлены в виде таблицы 1.

Из этих уже существующих данных необходимо создать выборку для обучения классификатора. Кроме того, нужно создать алгоритм сегментации изображения на полотно доски и фон. Вследствие комплексности рассматриваемой задачи, имеет смысл разбить ее на несколько более простых и узконаправленных подзадач.

1. Генерирование тренировочных данных и данных для тестирования. Для начала нужно сформировать данные в удобном для обучения и тестирования виде.

2. Выбор оптимального алгоритма машинного обучения. Исходя из специфики реализуемой задачи, необходимо выбрать наиболее подходящий классификатор, так как есть огромное множество фундаментально различающихся алгоритмов машинного обучения.

3. Тренировка классификатора на основе данных, выбранных для обучения. На этом шаге создается половина финальной модели, которая будет классифицировать полученные дефекты.

4. Тестирование полученного классификатора и анализ результатов. Промежуточная проверка реализованного классификатора покажет, является ли выбранный алгоритм оптимальным. Если же на тестовых данных он показал плохой результат, то мы сможем поменять его или перенастроить вовремя.

5. Создание подходящего алгоритма локализации методами компьютерного зрения. На этом этапе нужно решить сразу несколько совершенно разных задач, поэтому стоит разбить его на несколько пунктов:

   – Сегментация изображения на фон и полотно доски. Данный этап позволит нам определить первичную область интереса (полотно доски) и в дальнейшем рассматривать исключительно ее.

– Выделение дефектов на полотне доски. Данный этап локализует дефекты на полотне доски, образуя из дефектов уже необходимые для классификации области интереса.

6. Встраивание классификатора в модель компьютерного зрения. Необходимо объединить воедино полученные методы локализации полотна доски и дефектов и алгоритм-классификатор, который будет определять класс конкретного дефекта. Таким образом, получаеся цепочка следующих действий:

   - Модель принимает на вход изображение;
   - Модель отделяет фон от доски;
   - Модель локализует дефекты на полотне доски и передает массив полученных данных в классификатор;
   - Классификатор, встроенный в модель, идентифицирует к какому типу нужно отнести данный дефект;
   - Модель формирует выходное изображение, на котором выделены дефекты и им в соответствие поставлены типы.

7. Финальное тестирование прототипа. Последним этапом будет проверка созданного прототипа на изображениях, не вошедших ни в выборку для тренировки, ни в выборку для тестирования. Таким образом, мы сможем сформировать окончательные выводы об успешности и применимости данного прототипа.

Предполагается, что в результате решения задачи будет создана система, которая получает на вход необработанное изображение и автоматически его обрабатывает, локализуя и классифицируя обнаруженные дефекты.

2. Обзор методов

Для начала опишем задачи, стоящие перед нашей моделью и выберем методы, подходящие для решения задач, описанных в главе 1. Так как методов достаточно много, постараемся выбрать несколько фундаментально отличающихся алгоритмов и реализовать их, чтобы потом сравнить их работу на практике.

2.1. Визуальная система классификации

Для реализации визуальной системы классификации дефектов на пиломатериалах нам потребуется решить несколько задач.

Во-первых, задачу локализации области интереса или ROI[1], где в качестве ROI выступает пласть доски.

Во-вторых, задачу локализации дефектов на пласти, также выступающих интересующими нас областями (ROI).

В-третьих, задачу классификации найденных областей интереса (дефектов на пласти) и их разметка на выходном изображении.

После прочтения различной литературы, связанной с тематикой распознавания образов[15][14], и анализа прочитанного, для первой задачи было решено использовать преобразование Хафа для нахождения линий[2], а для решения последующих сравнить на практике методы нахождения особых точек (SIFT– и SURF–алгоритмы) в связке с классификацией линейным методом опорных векторов[3] и глубокую свёрточную нейронную сеть AlexNet вместе с такими методами компьютерного зрения, как фильтрация, математическая морфология изображения, пороговая обработка изображения и прочие. Перейдем к рассмотрению задач и каждого из методов, применяемых для их решения. Стоит также отметить, что разделение методов по задачам является условным,

[1]Region of Interest.

[2]Hough Line Transform

[3]Linear Support Vector Classification, Linear SVC

потому что некоторые из них использовались в ходе решения сразу нескольких задач.

2.2. Локализация пласти доски

Для нахождения пласти доски нам нужно найти ее границы. Для начала необходимо преобразовать исходное изображение, используя оператор Кэнни для определения границ, так как между фоном и границами доски имеется резкий контраст. Но обнаруженные границы могут содержать не только границы доски, но и границы других объектов. Поэтому необходимо каким-то образом отделить искомые границы доски от всех найденных оператором Кэнни границ.

2.2.1. Преобразование Хафа

Преобразование Хафа [6] является численным методом и алгоритмом для поиска объектов на изображении. Данный метод помогает обнаружить объект, если он представляет собой фигуру, которую можно описать математически (такие фигуры как линия, эллипс или круг). Кроме того, преобразование Хафа позволяет найти фигуру, которая была частично скрыта другими объектами или же искажена. Так как при помощи этого алгоритма мы ищем линии, то и рассмотрим простейший из случаев, линейное преобразование Хафа.

Любая прямая линия может быть представлена в виде $y = mx + b$ и вычислена в любой паре точек (x, y) на изображении. Лейтмотив преобразования Хафа заключается в том, чтобы учесть характеристики прямой в терминах ее параметров. Таким образом, прямая может быть представлена в пространстве параметов точкой (b, m), где b – это точка пересечения с осью OY, а m – коэффициент наклона.

Но параллельные оси ординат прямые имеют бесконечные значения для параметра m. Ввиду этого, удобнее выразить уравнение прямой при помощи параметров ρ и θ. В данном случае параметр ρ обозначает длину нормали, проведенной из точки отсчета до прямой, а θ – угол между этой нормалью и осью абсцисс. При использовании этих параметров

для описания уравнений прямых исключено возникновение бесконечных параметров. Используя данные параметры, уравнение прямой ли-

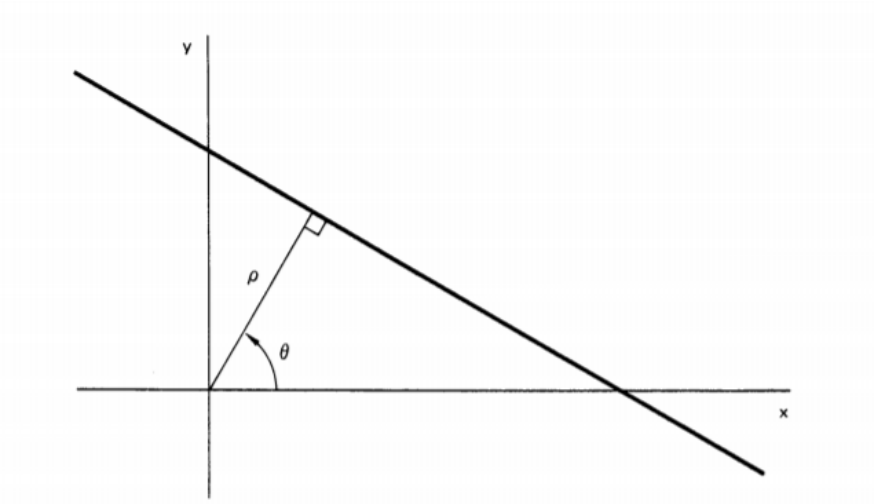


Рис. 2: Параметры ρ и θ на графике

нии можно записать как:

$$y = \left(-\frac{\cos(\theta)}{\sin(\theta)}\right) x + \left(\frac{\rho}{\sin(\theta)}\right), \tag{1}$$

А после преобразования это уравнение приводится к следующему виду:

$$\rho = x\cos\theta + y\sin\theta. \tag{2}$$

Поэтому для любой прямой на исходном изображении в двумерной плоскости, задаваемой стандартными осями абсцисс OX и ординат OY верно, что возможно связать ее с точкой (ρ, θ), лежащей в плоскости параметров, и она является уникальной, если $\theta \in [0, 2\pi]$ и $\rho \geq 0$ или $\theta \in [0, \pi]$ и $\rho \in \mathbb{R}$. Через любую точку плоскости прямых может проходить бесконечное множество. Пусть точка имеет координаты (x_a, y_a), тогда уравнение любой прямой, которая проходит через эту точку, задается как:

$$\rho(\theta) = x_a\cos\theta + y_a\sin\theta. \tag{3}$$

Это соответствует синусоидальной линии в пространстве Хафа (ρ, θ), которая, в свою очередь, единична для этой точки и однозначно её определяет. Когда эти кривые линии, соответствующие двум точкам, накладываются друг на друга, то точка (в пространстве Хафа), где они

пересекаются, соответствует прямым (в оригинальном месте изображения), которые проходят через обе точки. В общем случае, ряд точек, которые формируют прямую линию, определяют синусоиды, которые пересекаются в точке параметров для той линии. Таким образом, проблема обнаружения коллинеарных точек может быть сведена к задаче обнаружения пересекающихся кривых.
Нам известно, что сами границы доски всегда представляют собой горизонтальные прямые линии. Такие элементарные геометрические объекты возможно найти при помощи данного преобразования. По найденным линиям границ доски и будет производится отсечение фона.

2.3. Локализация дефектов на области доски

2.3.1. Сегментация изображений при помощи пороговой обработки

Можно заметить, что дефекты в большинстве случаев качественно отличаются по цвету от пласти доски. Поэтому представляется возможным определять области интереса на основе того, что искомые объекты темнее остальной плоскости доски. Соответственно можно предположить, что яркость областей дефектов будет лежать в несколько другом интервале, нежели, например, яркости годовых линий и пласти. Зная этот диапазон яркости, в котором существуют дефекты, можно будет провести пороговую обработку изображения. Пусть будет некоторый интервал (b_0, b_1), определяющий изменения яркости пикселей. В таком случае можно бинаризовать (4) изображение по двойному порогу.

$$f'(x,y) = \begin{cases} 0, f(x,y) \leq b_1 \\ 1, b_1 < f(x,y) < b_2 \\ 0, f(x,y) \geq b_2 \end{cases} \tag{4}$$

В том случае, если яркость пикселя принадлежит заданному диапазону, его значение приравнивается единице (на практике обычно 255, что равняется белому цвету), в иных случаях значение обращается в

0 и пикселю присваивается черный цвет. Порог так же может быть не двойным, а задаваться одной константой b. Если это верхний порог, то все значения больше b равны 255, значения меньше b равны 0. Для нижнего порога происходит в точности противоположное. Для того, чтобы определить необходимый диапазон изменения яркости, можно вычислять гистограмму яркости. Построение такого графика подразумевает распределение пикселей по уровням яркости. По этим уровням можно выбрать тот самый интервал, в котором содержатся значения яркости областей дефектов. По найденному интервалу можно произвести сегментирование изображения бинаризацией по порогу, и такое бинарное изображение будет содержать в себе контуры искомых дефектов.

2.3.2. Математическая морфология

Математическая морфология применяется как метод в анализе изображений, при котором какой угодно объект изображения выступает в качестве элемента некоторого пространства. Операции же осуществляются не над отдельными пикселями, а над элементами данного пространства. Относительно изображений, любая операция математической морфологии совершает преобразование входящего изображения P в новое изображение P_1. Начальными данными являются само изображение P и некий структурный элемент E. Структурным элементом является какое-то симметричное двоичное изображение произвольной формы с выделенным начальным элементом, который можно определить как геометрический центр. Выполнение операций математической морфологии есть ни что иное, как процесс сканирования изображения P при помощи структурного элемента E. Самыми популярными структурными элементами являются диск и квадрат. В данной работе мы используем две базовых операций математической морфологии, это расширение (dilation) (5) и сужение (erosion) (6), а также производную операцию открытия (7). Используя те же обозначения для структурного элемента и входного изображения, эти операции будут выглядеть следующим образом:

$$P \oplus E = \cup_{i \in P} E_i \tag{5}$$

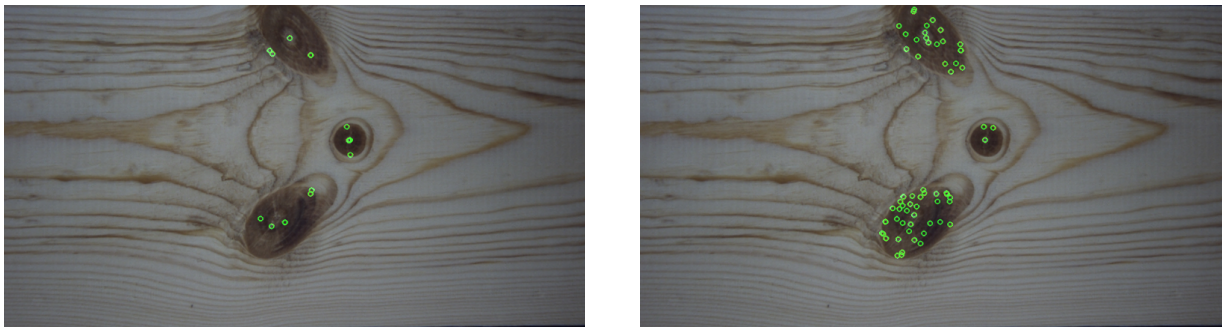
$$P - E = \{p : p + e \in P, \forall e \in E\} \tag{6}$$

$$P \circ E = (P - E) \oplus E \tag{7}$$

Как было установлено ранее, на определенных моментах будет производится работа с бинарными изображениями. Данные изображения могут содержать шум и неточные контуры, а для их устранения возможно применить методы математической морфологии. Поэтому они потребуются в процессе реализации методов локализации.

2.3.3. Алгоритмы нахождения особых точек и построения дескрипторов

Применение алгоритмов нахождения особых точек и построения дескрипторов было обусловлено тем, что таким образом получив инвариантность на довольно большое количество преобразований, таких как смещения, поворот, масштаб, изменения локации камеры и яркость. Кроме того, данные особые точки и сопутствующие им дескрипторы можно вычислить на областях интереса. Следовательно, для каждого дефекта, в который будут входить найденные точки и дескрипторы, можно сформировать массив данных. А на его основе будет удобно обучить Linear SVC, в дальнейшем передавая в него только особые точки и дескрипторы от вычисленных областей интереса для классификации типа дефекта. На изображении (рис. 3) приводится пример вычисления ключевых точек с использованием областей интереса:



а) Вычисленные точки по алгоритму SIFT

б) Вычисленные точки по алгоритму SURF

Рис. 3: Примеры особых точек

Опытным путем было установлено, что SURF–алгоритм справляется со спецификой задачи несколько лучше, чем SIFT–алгоритм, так как на большой выборке изображений он обнаружил большее количество точек в областях дефектов. Подробное описание работы данных алгоритмов экстракции особых точек можно найти в этих работах — для Scale Invariant Feature Transform–алгоритма [5] и для Speeded Up Robust Features-алгоритма [9].

2.4. Классификация дефектов

Расмотренные далее методы фундаментально отличаются друг от друга. В первую очередь, они отличаются типами данных, которые нужно сформировать для их обучения, и которые в дальнейшем нужно сгенерировать для распознавания областей интереса. Поэтому для их реализации будет необходимо построить две совершенно разные модели, основанные на разных методах.

2.4.1. Linear SVM Classifier

В целом, задача классификации состоит в том, чтобы определить, к какому классу принадлежит объект. Таким объектом обычно является вектор в n-мерном вещественном пространстве $\mathbb{R}$. Если классов всего два, то речь идет о решении задачи бинарной классификации. Но в поставленной перед нами задаче 11 классов, поэтому необходимо решить задачу мультиклассовой классификации. В общем виде ее можно сформулировать следующим образом, пусть X - пространство объектов, а Y - вектор классов (например, Y = {0, 1} или Y={яблоко, апельсин, груша}). Классификатор же будет некоторой функцией $F : X \to Y$, однозначно сопоставляющей класс y объекту x. Одним из методов решения подобной задачи является метод опорных векторов, подробнее о котором можно прочитать здесь [12].

В ходе реализации задачи мы применим Linear SVM Classifier. Он является оптимальным методом для решения этой задачи в силу того, что мы обладаем сравнительно небольшой выборкой для обучения класси-

фикатора.

В рамках нашей задачи не представляется возможным передавать в классификатор, основанный на методе опорных векторов, изображения дефектов в качестве объектов. Поэтому для ее решения нужно сначала определить признаки каждого из дефектов. Признаками, или особенностями, как было сказано ранее, будут выступать особые точки и их дескрипторы, вычисленные SIFT– или SURF–алгоритмами. Таким образом, для каждой установленной области интереса мы вычислим ключевые точки и дескрипторы, на основе которых обучим данный классификатор. Для классификации дефектов обученным классификатором нужно будет локализовать дефекты и вычислить их ключевые точки и дескрипторы, которые и будут передаваться в классификатор как объект x, для которого классификатор определит класс y.

Но, как оказалось в ходе реализации подобного прототипа, построенная модель показало неудовлетворительную точность (36% успешного определения) на тестовых данных. Для вынесения конечного решения о перспективности данного подхода были сформированы новые данные для обучения, которые содержат в себе меньшее количество классов, 4 против 11 изначальных, за счёт объединения подобных классов (сучков). Но и на данной выборке классификатор показал всего 64% успешности определения класса, что является незначительным приростом для подобной существенной аппроксимации. Поэтому было решено не усложнять методы извлечения особенностей областей интересов, а в целом поменять алгоритм машинного обучения.

2.4.2. Глубокие свёрточные нейронные сети : AlexNet

В качестве следующего подходящего для решения задачи классификатора была выбрана глубокая свёрточная нейронная сеть, или CNN[4]. Если стандартным алгоритмам машинного обучения для распознавания образов необходимо сначала извлечь какие-либо признаки, пользуясь классическим подходом, будь то работа с градиентом изображения,

[4]Convolutional Neural Network

поиск ключевых точек и т.д., то CNN самостоятельно производит похожую работу.

Вообще говоря, свёрточные сети не так сильно отличаются от привычных нейронных сетей. Свёрточная сеть тоже состоит из нейронов, имеющих свой собственный вес, который может меняться. Каждому такому нейрон отправляются некоторые входящие данные, и он совершает над ними скалярное произведение. Сеть целиком все так же можно представить в виде дифференцируемой функции, на вход она получает необработанное изображение, как входной вектор данных, а на выходе дает оценку класса данного изображения. Для последнего fully-connected слоя возможна реализация функции потерь, такой как Softmax или SVM. Этот последний полностью связанный слой называют выходным, потому что именно на нем классифицируется входное изображение. Также с обычными сетями их роднит то, что многие методы обучения обычных нейронных сетей справедливы и могут использоваться для свёрточных сетей.

Свёрточные нейронные сети отличаются тем, что сама архитектура такой сети явно предполагает использование изображений в качестве начальных данных. Данное условие помогает закодировать некоторые свойства в архитектуру. Таким образом, можно задавать создание сети более эффективным путем, при этом сокращая количество параметров. Свёрточные нейронные сети получают входные данные в виде одного вектора и преобразуют их, пропуская через скрытые слои. Каждый скрытый слой построен из некоторого множества нейронов. Нейрон такого слоя полностью связан со всеми нейронами предыдущего слоя, но не связан с другими нейронами своего слоя и функционирует независимо от них.

Свёрточные сети решают проблему плохой масштабируемости обыкновенных сетей при работе с изображениями большого размера. В отличие от обычной нейронной сети, слои CNN состоят из нейронов, расположенных в 3-х измерениях: ширине, высоте и глубине, то есть измерениях, которые формируют объем (рис. 4). Поэтому каждый слой свёрточной сети производит трансформацию входного трехмерного объема

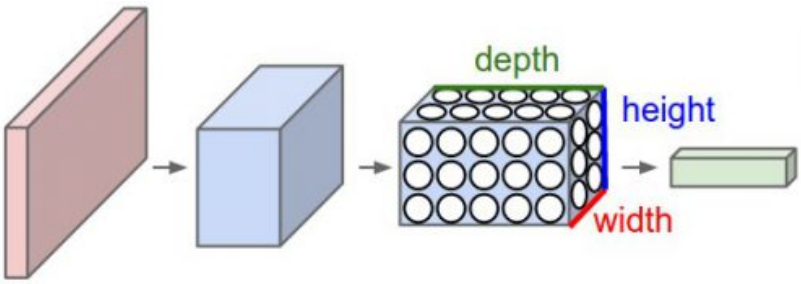


Рис. 4: Структура CNN

в выходной трехмерный объем с некоторой функцией активации. Некоторая простая архитектура свёрточной сети может состоять из пяти разных слоев: Input, Conv, ReLU, Pool, FC. В Input-слое хранится начальная информация об изображении как $[w \times h \times c]$, где w и h – это ширина и высота, c – глубина, определяемая количеством цветовых каналов (например, это могут быть 3 канала цветовой модели RGB).

Слой Conv означает свёрточный слой (от convolutional), здесь производится свёртка исходного изображения фильтром. Фильтр обычно кодирует какой-то признак и является какой-то матрицей, поэтому происходит накладывание фильтра на изображение всеми возможными способами, производится поэлементное умножение и сумма этих произведений записывается как элемент новой матрицы. Глубина получаемого объема зависит от количества фильтров на слое, например, при 15 фильтрах получится объем $[w \times h \times 15]$.

Слой ReLU, или же блок линейной ректификации, реализует простой пороговой переход в нуле, как функция вида $f(x) = max(0, x)$. Эта функция позволяет в несколько раз уменьшить время обучения по сравнению с традиционным гиперболическим тангенсом.

Pool-слой, также слой субдискретизации или слой подвыборки, уменьшает размеры объема по ширине и высоте при помощи понижающей дискретизации. Если предшествующая свёртка определила некоторые признаки, то для дальнейших действий такое подробное изображение не нужно и оно уплотняется до меньшего размера.

FC-слой, он же fully–connected слой, обращается к выводу предыдущего слоя, определяет характерные для каждого класса свойства и для классификации выводит n-мерный вектор, где n - это количество классов.

Процесс обучения сети является итерируемым, одна итерация называется эпохой обучения, которая состоит из обучения сети на всей тренировочной выборке и контрольной оценки точности. Таких эпох обычно производится много, и потом выбирается лучшая из них.
Для реализации задачи была использована свёрточная сеть AlexNet [4]. В оригинальной статье сеть была обучена на данных ImageNet. В качестве функции активации здесь выступает ReLU. При помощи смещений, горизонтальных отражений, выделения фрагментов и иных преобразований исходных изображений была реализована аугментация данных[5]. Что типично для свёрточных нейронных сетей, AlexNet имеет большое количество параметров, что может приводить к переобучению[6]. Для предотвращения переобучения в данной модели был реализован dropout [2], один из методов регуляризации. Обучение было выполнено с помощью пакетного стохастического градиентного спуска (batch stochastic gradient descent) при фиксированных значениях импульса (momentum) и коэффициента затухания весов (weight decay). Данный алгоритм появился еще в 2012 году, и за это время сеть AlexNet успела себя зарекомендовать как один из лучших классификаторов для задач с с более чем двумя классами, не сильно потеряв в актуальности за прошедшие пять лет (методы, используемые в ней, до сих пор применяются в свёрточных нейронных сетях). Всего в ней (рис. 5) 8 слоев: 5 свёрточных и 3 полностью связанных, 60 миллионов параметров и 650 тысяч нейронов.

Модель AlexNet, использовавшаяся в данной работе, была реализована с Softmax функцией потерь на выходном слое.

[5]Создание дополнительных данных

[6]Переобучение (переподгонка, англ. overfitting) – явление, когда нейронная сеть слишком хорошо адаптируется к конкретным условиям обучающей выборки и теряет способность к обобщению и экстраполяции. Это отражается в плохой классификации образцов, не вошедших в выборку для обучения.

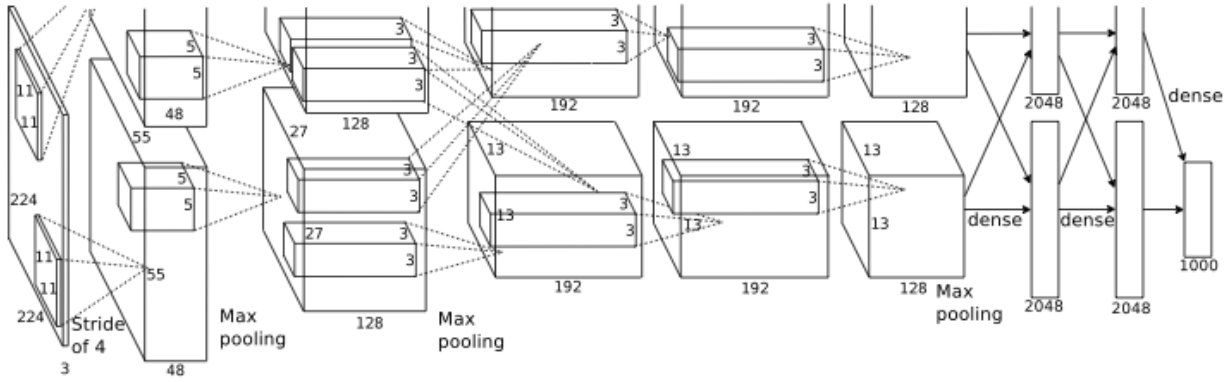


Рис. 5: Архитектура AlexNet. Архитектура разбита на два потока, потому что была обучена на двух видеокартах.

3. Реализация задачи

В качестве языка программирования для разработки был использован Python, при реализации первой модели, основанной на методе опорных векторов, использовались библиотека алгоритмов компьютерного зрения OpenCV и библиотека машинного обучения Scikit-Learn. Для реализации второй модели снова потребовалась библиотека OpenCV, для построения классификатора - библиотека TensorFlow для работы с нейронными сетями. После изучения сопутствующих материалов[10], обучение производилось на видеокарте, так как это существенно увеличит скорость процесса. Для осуществления обучения на GPU необходимы пакеты NVidia CUDA и NVidia CUDnn. Вся работа осуществлялась на ноутбуке с данными характеристиками:

Центральный процессор(CPU)	Intel Core i7-4702MQ 2.2 GHz
Оперативная память	8 GB DDR3 1600 МГц
Видеокарта(GPU)	NVIDIA GeForce GTX 850M 2 GB

3.1. Общая архитектура

Обе созданные модели работают по одному общему принципу, описанному на схеме (рис. 6). Данная архитектура является лишь обобщенным описанием работы. Кроме того, после этапа удаления фона методы локализации дефектов, имплементированные в модели, начинают раз-

личаться. Сейчас рассмотрим работу моделей в общих чертах, а далее уже раскроем каждый этап и его методы более подробно.

Рис. 6: Обобщенный алгоритм построенных моделей

Для начала работы нужно установить камеру над конвеером, по которому проезжают доски и центрировать их на получаемых фотографиях (или же добиться хотя бы того, что доска появляется в кадре целиком). После того, как алгоритм получил изображение, он начинает поиск пласти доски на изображении. Для этого он преобразует цветное изображение из цветовой модели RGB в одноканальное grayscale[7] изображение. Переход к градациям серого осуществляется по следующей формуле:

$$Y' \leftarrow 0.299 \cdot R + 0.587 \cdot G + 0.114 \cdot B \tag{8}$$

Этот переход позволяет применить необходимые алгоритмы для отсечения фона, а изображение получает новый размер, преобразуясь в изображение только пласти доски. Данное изображение обрабатывается алгоритмами нахождения контуров, которые локализуют дефекты как области интереса на плоскости доски. На этом этапе применяются различные фильтрации изображения. На основе их работы создается массив данных, каждый элемент которого является отдельно взятой

[7]Оттенки серого (градации серого, шкала серого цвета, англ. Grayscale) — цветовой режим изображений, которые отображаются в оттенках серого цвета, размещённые в виде таблицы в качестве эталонов яркости белого цвета. Чаще всего используют ступенчатое изображение равномерного ряда оптических плотностей нейтрально-серых полей.

областью интереса из всех обнаруженных ROI. Данный массив передается в алгоритм классификации для определения типа дефекта на области интереса. После вынесения классификатором вердикта, дефект, в зависимости от данного ему класса, размечается кривыми линиями определенного цвета, которые в совокупности образуют полигон. Выходное изображение представляет собой копию начального, но с нарисованными на нем цветными многоугольниками, описывающими дефект и таблицей соответствия цветов линий на изображении и классов дефектов.

3.2. Выделение границ доски

Как уже было сказано, для работы этого алгоритма нужно перевести изображение в градации серого, после чего к нему можно применить оператор Кэнни. К полученному изображению найденных границ применяется операция расширения, базовая операция математической морфологии, результат отображен на рис. 7. Такое действие увеличит размеры обнаруженных оператором Кэнни границ и позволит нам успешно применить линейное преобразование Хафа на любом изображении.

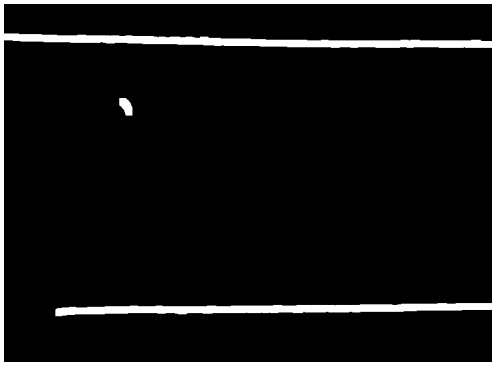


Рис. 7: Применение оператора Кэнни и расширения

Для нахождения границ пласти совершим линейное преобразование Хафа со значениями $\rho = 1$ и $\theta = \frac{\pi}{2}$, эти параметры позволят нам найти только горизонтальные линии. Для определения пласти нужно от параметров ρ и θ и их пространства перейти обратно в прямоугольную систему координат и найти максимальное и минимальное значение y_{max} и y_{min} из всех возможных y. После этого просто обрежем изображение по

высоте, иначе говоря по оси OY, в диапазоне $(y_{min} + c, y_{max} - c)$, где c - это некоторая константа, обеспечивающая небольшой отступ от границ внутрь пласти доски. Обычно $c = 10$, что значит отступ в 10 пикселей. Это сделано из-за того, что на изображениях досок присутствует прямая линейная перспектива, в следствие которой границы доски не параллельны оси OY, по которой производится преобразование изображения. Поэтому в преобразованном изображении может присутствовать фон, а это может повлиять на дальнейшую работу прототипа. На рис. 8 белыми линиями изображены найденные границы доски.

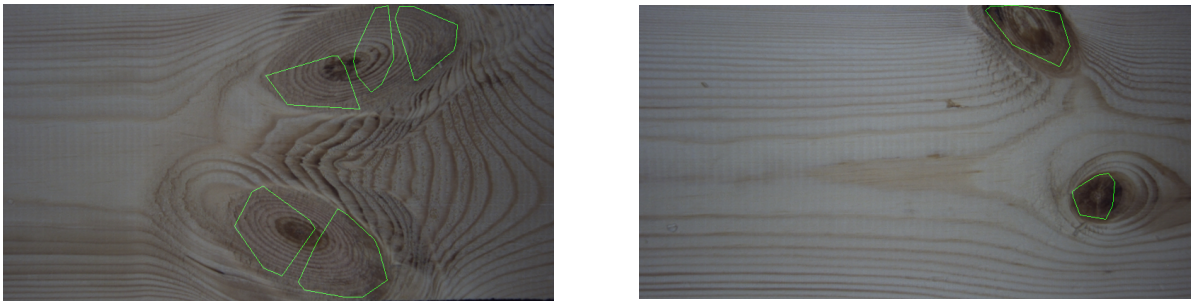
Рис. 8: Результат применения `cv2.HoughLines()`

3.3. Выделение контуров

В ходе реализации было создано три разных алгоритма нахождения контуров дефектов. На основе этих контуров можно сформировать изображения только дефектов без пласти доски, что и является нашими областями интересов. Был проведен сравнительный анализ двух разработанных методов и из них был найден оптимальный для решения задачи.

1 метод: Кластеризация ключевых точек. Взяв изображение пласти доски, найдем на нем особые точки SURF–алгоритмом. Логично, что количество особых точек на дефектах будет больше, потому что

они качественно отличаются цветом и формой от остальной плоскости доски и годовых линий. При помощи метода распространения близости (Affinity Propagation[1]), кластеризуем найденные особые точки. От этих кластеров точек вычислим выпуклую оболочку[13]. Таким образом, мы получили контуры искомых дефектов.



а) Плохая локализация б) Хорошая локализация

Рис. 9: Пример обнаружения дефектов

Плюсы данного метода заключаются в том, что он помогает распознавать разные классы дефектов, имеющих серьезные отличия в цвете и форме. То есть, гипотетически, его можно одинаково эффективно применить и к локализации таких типов дефектов, как сучки (имеют более темный оттенок по сравнению с пластью), и к нахождению механический вмятин, которые статистически имеют более светлый оттенок. Но при проверке данного метода выяснилось, что он обладает не высокой точностью в большинстве случаев, и лишь в некоторых, особо жестких условиях может показывать приличный результат. Таким образом, этот минус будет перекрывать любые его плюсы. Если заняться его модификацией, возможно расширить границы условий, в которых он будет хорошо локализовывать области интереса, но проще и логичнее было реализовать другой метод нахождения дефектов.

2 метод: Экстракция областей интереса фильтрацией на основе яркости объекта. На вход методу аналогично подается изображение плоскости доски, после чего приводится к оттенкам серого. Для уже серого изображения вычислим гистограмму яркости изображения, в каждой корзине которой лежат пиксели соответствующего интервала

яркости. Мы исходим из той логики, что пиксели дефектов будут систематически темнее пикселей пласти и годовых линий, но их будет меньше. На сером же изображении, яркость пикселей в области дефектов будет несколько меньше яркости остальной плоскости доски. Очевидно, что корзина, содержащая пиксели пласти и годовых линий, будет наибольшей, потому что все таких пикселей на изображении доски будет больше всего. Поэтому, взяв номер корзины ind, следующей за корзиной с самым большим количеством пикселей, мы сможем сегментировать на основе пороговой обработки нужные нам области, которые находятся в диапазоне $(0, step \cdot ind)$, где $step$ - константа, равная 26. Благодаря такой пороговой обработке получается бинарное изображение, где белым выделены области дефектов, а все остальное - черным. Но стоит учитывать, что темные дефекты обычно не имеют равномерный темный цвет. Поэтому внутри обнаруженной белой области может присутствовать большое количество черных пикселей, что сделает невозможным нахождение контура целиком. Для получения необходимых полностью белых областей снова используется расширение. С помощью полученных областей интереса можно точно определить контуры дефектов. По этим контурам впоследствии можно создавать бинарные маски. Если побитово умножить подобную маску с изображением доски, мы получим изображение только дефекта на черном фоне вместо плоскости доски. Такое изображение уже можно передавать в классификатор. Локализация дефектов данным методом изображена на рис. 10.

Рис. 10: Локализованные дефекты

Самым весомым плюсом, особенно по сравнению с предыдущим методом, будет то, что данный метод прост и эффективен. При проверке на большом количестве изображений он показал уверенное нахождение темных дефектов. Но его однозначным минусом будет то, что он был создан и работает только для дефектов более темного цвета. Для механических повреждений он будет работать лишь в том случае, если при освещении доски вокруг данного класса дефекта образовалась тень, что позволит его занести в области интереса этим методом. Поэтому был реализован третий метод, который является дополнением второго и устраняет данный пробел.

3 метод: Нахождение механических дефектов. Как уже было сказано ранее, механические вмятины имеют более светлый оттенок или же такой же, что и сама плоскость доски. Поэтому в отношении данного класса дефектов второй метод работает лишь частично, потому что при вычислении гистограммы изображения механические повреждения довольно часто оказываются в одной с пикселями пласти и годовых линий корзине. Соответственно, для их локализации нужен иной подход. В начале уже привычно преобразуем входящее изображение в grayscale изображение. Для того, чтобы лучше выделить механические дефекты, необходимо увеличить контрастность изображения. Для этого применим метод CLAHE[8].

Суть этого метода[11] заключается в том, что изображение разбивается на части, и в каждой части вычисляется своя гистограмма распределения яркости. Полученные гистограммы используются для перераспределения значений светлости. Таким образом, увеличивается контрастность, что позволяет обнаруживать плохо различимые детали изображения и лучше определять грани. Но одновременно с этим наблюдается закономерность перенасыщения на изображении монотонных областей. Для того чтобы предотвратить это чрезмерное усиление шума, к каждой части изображения применяется контрастное ограничение.

[8]Contrast Limited Adaptive Histogram Equalization - констрастно-ограниченная адаптивная эквализация гистограммы

После применения этого метода на изображении необходимо инвертировать его. Обработаем при помощи порога полученное изображение, оставив только пиксели, лежащие на небольшом интервале значений, используем на нем открытие, операцию математической морфологии (она заключается в применении расширения сразу после сужения), взяв в качестве структурного элемента эллипс. После совершения данной операции, размоем изображение благодаря последовательному использованию гауссовской пирамиды, сначала увеличивая его размер, а потом понижая до исходного. Если снова применить пороговую обработку с многократно увеличенным диапазоном на изображении, а потом совершить на нем операцию открытия, то мы получим изображение, на котором можно определить объекты, похожие на блобы[9]. Некоторые из таких блобов содержат искомые механические вмятины. Для их локализации используем blob-детектор. Данный детектор создает из исходного бинарные изображения, применив пороговую обработку от заданного минимального порога до максимального с некоторым шагом. После этого находит контуры связанных компонентов и находит точку их центра. Эти точки центров со всех бинарных изображений группируются по координатам. Близкие к друг другу точки образуют группу, соответствующую одному блобу. Из данных групп можно вычислить конечный центр блоба и его радиус, которые передаются в виде особых точек. Поэтому, использовав его на изображении с определенными порогами, мы можем найти блобы механических вмятин, получить особые точки и радиусы блобов. По этим данным искомые дефекты можно локализовать на пласти доски.

Итак, получив контуры дефектов от второго и третьего методов, объединим их в один массих данных, получив все контуры дефектов, нужные для классификации. Создав темное изображение тех же размеров, что и входное, заполним на нем внутренности контуров белым цветом, таким образом получив бинарную маску. Произведя побитовое умножение данной маски на входное изображение, мы создадим нужное нам

[9]В данном случае блобом(англ. blob) будет называться каплеобразный регион на изображении, чья яркость или цвет будут сильно отличаться от яркости/цвета окружающих регионов.

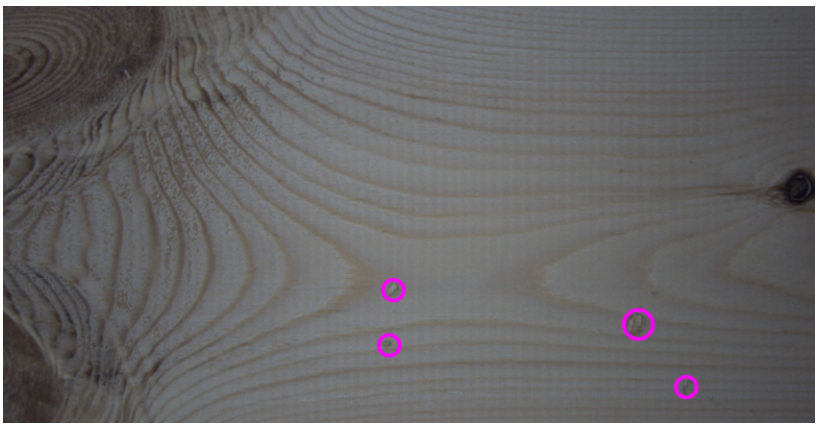


Рис. 11: Пример обнаружения механических дефектов

изображение только дефектов без учета пласти доски. Реализация приводится в листинге 1.

3.4. Классификация дефектов

Для применения любого алгоритма машинного обучения, сначала необходимо сгенерировать данные подходящего вида. Изначально наши данные имеют довольно сложную структуру и их было бы сложно применять в первозданном виде. Поэтому потребовалось реализовать программу, которая брала из исходных данных изображения досок, типы дефектов и их контуры на плоскости доски, после чего образовывалась новые данные, в которых каждому изображению дефекта, отделенного от исходной фотографии доски, ставился в соответствие класс данного дефекта. Такое преобразование данных серьезно упрощает все дальнейшие манипуляции с изображениями. Стоит также отметить, что для обучения стандартных алгоритмов машинного обучения в коллекцию нужно сохранять не сами изображения дефектов, а извлеченные из них признаки.

Так как в ходе промежуточных проверок модель, основанная на классификаторе с линейным методом опорных векторов показала неудовлетворительные результаты, подробно рассматривать ее реализацию мы не будем. Можно предположить, что модель имеет право на жизнь в том случае, если для ее обучения и впоследствии для классификации будет применятся более тонкий и сложный метод экстракции призна-

ков из областей интереса. Поэтому перейдем к рассмотрению модели с AlexNet в качестве классификатора.

3.4.1. Обучение сети AlexNet

Для начала отметим, что для обучения сети AlexNet для решения нашей задачи нам не нужно тренировать сеть целиком, будет вполне достаточным применить принцип fine-tuning[10]. Смысл довольно прост: взяв готовые веса, полученные после обучения сети на каком-либо большом наборе данных (например, ImageNet), вставим их в нашу модель, в целом получая готовую сеть для классификации объектов, аннотированных в ImageNet. После того, как мы распределили взятые веса по сети, мы производим тонкую настройку нескольких последних слоев, т.е. отбросив какое-то количество слоёв, перенастраиваем эти слои, используя собственный набор данных. Веса при этом оптимизируются дискретным методом обратного распространения ошибки. Таким образом, из сети, которая, допустим, использовала готовые веса от ImageNet и была готова распознавать тысячу с лишним классов изображений, определенных в данном наборе данных, мы получаем сеть, распознающую уже классы нашей задачи. Применение данного метода было обусловлено тем, что создание такой модели происходит намного быстрее, а в итоге получается результат, несущественно отличающийся в точности от полноценного обучения нейронной сети.

Для обучения и тестирования необходимо поделить сгенерированные ранее данные на выборку для тренировки с 70% всех изображений и тестовую выборку с 30% оставшихся изображений. Такое разбиение является оптимальным и помогает решить проблему переобучения.

В отличие от стандартных методов машинного обучения, которым в качестве данных нужны признаки классов, свёрточные нейронные сети в качестве данных используют изображения, самостоятельно определяя признаки классов. Входные данные подаются в виде пакета, который состоит из некоторого количества изображений. Для создания такого

[10]Тонкая настройка

пакета изображений используется программа `datagenerator.py`, описанная в листинге 5, которая преобразует массив входных изображений в пакеты из 128 изображений установленного в сети размера 227x227x3. При желании эти два параметра возможно изменить. Схему обучения данной модели можно представить в следующем виде:

1. Создается сеть AlexNet (листинг 3), в нее загружаются готовые веса;

2. В модель загружаются файлы *train.txt* и *test.txt*, содержащие в себе информацию об обучающей и тестовой выборке;

3. Из выборок генерируются пакеты по 128 изображений размера 227x227x3;

4. На основе этих пакетов производится fine-tuning (листинг 4) трёх последних полностью связанных слоев: [fc6, fc7, fc8];

5. Сеть производит оценку точности, сохраняет эпоху и записывает лог.

Для упрощения наблюдения за обучением использовалось веб-приложение TensorBoard. После 500 эпох обучения, данная модель показала точность (рис. 12) в 98%, что является очень хорошим, конкурентоспособным результатом.

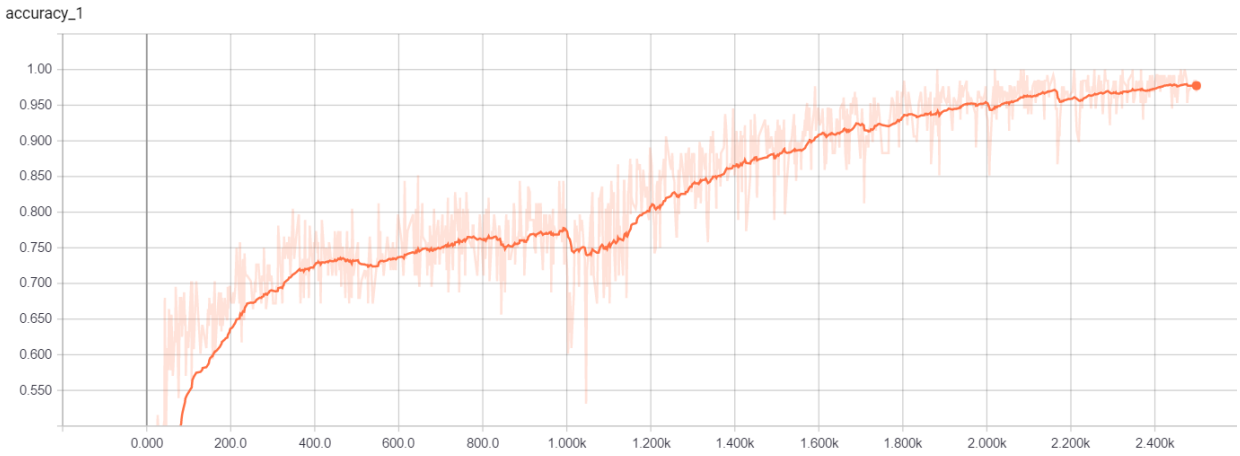


Рис. 12: Точность классификатора на тестовых данных

3.5. Объединение моделей локализации и классификации

Итак, мы получили эффективную модель классификации для решения нашей задачи. Данная сеть в качестве входного X берет массив изображения, преобразует его и передает на выход вектор Y с 11 значениями. Так как на выходном слое используется Softmax, каждое значение вектора Y означает вероятность присутствия класса на данном изображении. Поэтому в качестве итогового вердикта мы просто возьмем класс с максимальным значением в векторе Y. Но для того, чтобы получить финальную модель, нужно объединить модель локализации дефектов и модель классификации. Стоит отметить, что из-за того размера пакета, который был выставлен нами раньше, данная сеть может передает на выход не один вектор Y, а матрицу размерностью 128x11. То есть, за одну сессию классификации можно определять классы на 128 изображениях одновременно. Так или иначе, при локализации дефектов на доске чаще всего их оказывается больше одного, поэтому возможность передачи изображений в виде пакета является существенным плюсом, помогая единовременно распознать все дефекты на доске и положительно влияя на производительность в общем. Таким образом, для построения финальной модели остаётся только лишь напрямую передавать массив обнаруженных дефектов из методов локализации в классификатор и обеспечить вывод результатов на экран. Один из примеров выходных изображений демонстрируется на рис. 13.

После реализации (листинг 2) данной модели провелось финальное тестирование, благодаря которому стало ясно, что подобная система полностью выполняет требования задач локализации и классификации. Таким образом, созданная система является подходящим решением для задачи, поставленной в рамках данной работы.

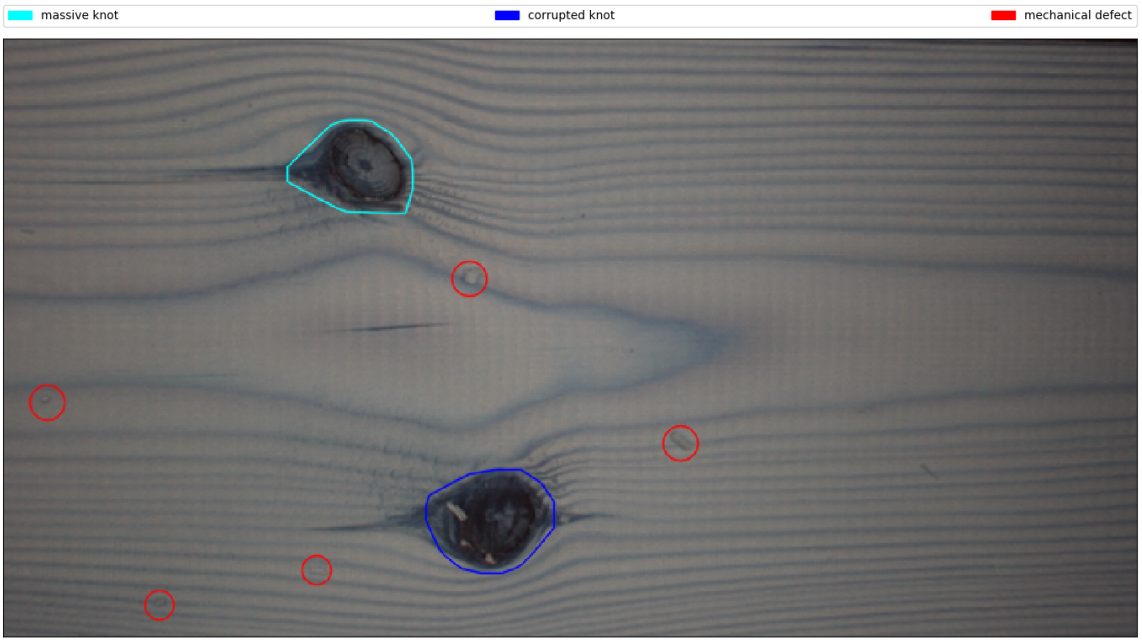


Рис. 13: Результат работы финальной модели

4. Выводы

Для локализации пласти доски и отсечения фона был создан алгоритм компьютерного зрения, пригодный для решения данной задачи. Для локализации дефектов на практике был опробован метод кластеризации особых точек, который оказался приемлем далеко не во всех случаях. Зато отличный результат показал составной метод нахождения областей интереса, обнаруживающий на досках все интересующие нас в рамках задачи объекты. В любом случае, хоть данный метод и является достаточным для решения поставленной задачи, его возможно дополнительно улучшить для того, чтобы он мог стать решением для более широкой подобной задачи.

Результаты первой модели, основанной на методе опорных векторов являются неудовлетворительными, всего 36% для 11 классов и 64% для 4, поэтому она была забракована на этапе промежуточных тестирований. Гипотетически, данная модель может показать хорошие результаты, но в таком случае необходима фундаментальная реновация всей модели. Для этого необходимо создать более искусный метод извлечения признаков из объекта интереса, возможно применяя сразу несколько

методов экстракции особенностей, генерировать на их основе выборку и на ней обучать данную модель. Возможным улучшением также может выступить реализация ансамбля алгоритмов машинного обучения, объединение сразу нескольких классификаторов для более точного решения задачи.
Модель, основанная на свёрточных сетях, показала свою успешность во время промежуточных испытаний, дав точность в 98% успешных предсказаний, и была встроена в прототип системы автоматической локализации и классификации. В итоге получилась модель, обеспечивающая полное решение поставленной задачи.

4.1. Возможное улучшение финальной модели

Для расширения возможностей осуществленной модели можно, опять же, совершенствовать существующий метод локализации. Если получить новые данные для обучения сети, таким образом расширив количество образцов, на которых сеть обучается, классификатор станет более точным и вариативным.
С другой стороны, можно рассмотреть переход от использования нейронных сетей для классификации к использованию Fully Convolutional Networks, так называемых свёрточных сетей. Данный вид сетей тоже выполняет задачу классификации, но их весомым отличием является то, что они к тому же способны выполнять задачу сегментации, что значит самостоятельное определение локации объектов. Нейронные сети могут выполнять такую задачу, если изменить подход к обучению – через всю сеть проходят необработанные изображения с объектами на них, а на последние слои подаются изображения самих объектов. Таким образом, сеть сама учится выделять объект на изображении. К FCN возможно привести и сеть AlexNet[8], но это потребует значительной переработки и добавления новых слоев.
Так как количество данных является сравнительно небольшим, одним из лучших методов решения подобной задачи будет полностью свёрточная сеть U-Net[7], применяемая в биомедицине для сегментации уль-

тразвуковых изображений. Но данная сеть в оригинале не сможет решить поставленную задачу, потому что имела на последнем выходном слое содержится сигмоида, что значит применимость оригинальной сети только для решения задач бинарной классификации. Поэтому для ее имплементации в модель необходимо будет произвести замену сигмоиды на Softmax или SVM, после чего произвести все производные изменения в структуре сети, либо совершить еще более сложную модификацию[3]. В теории, данный переход при грамотном применении позволит получить модель для решения более обобщенной задачи, без необходимости в тонкой настройке.

Заключение

Задача, поставленная в рамках данной выпускной квалификационной работы, состояла в создании системы локализации и классификации дефектов на пиломатериалах. Для этого подразумевалось решение сразу нескольких задач: отсечение фона от пласти доски, локализация дефектов и последующая их классификация. В ходе ее решения были получены следующие результаты:

1. Исследованы методы компьютерного зрения, необходимые для локализации пласти доски и отсечения фона.

2. Был осуществлен анализ существующих методов обнаружения особых точек и дескрипторов.

3. Был осуществлен анализ методов пороговой обработки, фильтрации изображения, морфологии и прочих для решения задачи локализации дефектов.

4. Построена модель локализации дефектов по кластеризации ключевых точек, для эффективного применения нуждающаяся в переработке.

5. Построен составной алгоритм локализации дефектов, доказавший свою эффективность на практике.

6. Построен классификатор, основанный на линейном методе опорных векторов. Был признан негодным в результате промежуточных испытаний.

7. Построен классификатор, основанный на свёрточной нейронной сети AlexNet. Данная система классификации показала точность в 98% и была включена в финальную модель.

8. На основе методов локализации и классификатора была разработана и осуществлена финальная модель, система автоматической локализации и классификации дефектов на пласти доски.

9. Было произведено тестирование получившейся системы и показана ее работоспособность.

10. Были предложены возможные улучшения забракованных методов, как и идеи оптимизации осуществленной системы, в частности при помощи FCN.

Исходя из всего вышесказанного, можно утверждать, что поставленная задача была решена полностью. Примененные в модели методы локализации показали целесообразность своего применения, был создан точный классификатор для определения типов дефектов, также реализована пакетная передача данных, что оптимизирует скорость работы. Кроме того, существующую модель при помощи предложенных вариантов улучшения в будущем возможно модифицировать для решения более обширной задачи.

Список литературы

[1] Brendan J. Frey, Delbert Dueck. Clustering by Passing Messages Between Data Points. — 2007. — URL: http://www.icmla-conference.org/icmla07/FreyDueckScience07.pdf.

[2] Dropout: A Simple Way to Prevent Neural Networks from Overfitting / Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky et al. — 2014. — URL: https://goo.gl/rZMHIF.

[3] Fully Convolutional Architectures for Multi-Class Segmentation in Chest Radiographs / Alexey A. Novikov, David Major, Dimitrios Lenis et al. // CoRR. — 2017. — Vol. abs/1701.08816. — URL: http://arxiv.org/abs/1701.08816.

[4] Krizhevsky Alex, Sutskever Ilya, Geoffrey E. Hinton. ImageNet Classification with Deep Convolutional Neural Networks. — 2012. — URL: http://www.cs.toronto.edu/~fritz/absps/imagenet.pdf.

[5] Meng Yu. Implementing the Scale Invariant Feature Transform(SIFT) Method. — 2008. — URL: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.102.180&rep=rep1&type=pdf.

[6] Richard O. Duda, Peter E. Hart. Use of Hough Transoformition to detect lines and curves in pictures. — 1971. — URL: http://www.ai.sri.com/pubs/files/tn036-duda71.pdf.

[7] Ronneberger Olaf, Fischer Philipp, Brox Thomas. U-Net: Convolutional Networks for Biomedical Image Segmentation // CoRR. — 2015. — Vol. abs/1505.04597. — URL: http://arxiv.org/abs/1505.04597.

[8] Shelhamer Evan, Long Jonathan, Darrell Trevor. Fully Convolutional Networks for Semantic Segmentation // CoRR. — 2016. — Vol. abs/1605.06211. — URL: http://arxiv.org/abs/1605.06211.

[9] Speeded-Up Robust Features (SURF) / Herbert Bay, Andreas Ess, Tinne Tuytelaars, Luc Van Gool. — 2008. — URL: https://goo.gl/cKxabb.

[10] Strigl Daniel, Kofler Klaus, Podlipnig Stefan. Performance and Scalability of GPU-based Convolutional Neural Networks. — 2010. — URL: https://goo.gl/eIjHcA.

[11] Zuiderveld Karel. Graphics Gems IV / Ed. by Paul S. Heckbert. — San Diego, CA, USA : Academic Press Professional, Inc., 1994. — P. 474–485. — URL: http://dl.acm.org/citation.cfm?id=180895.180940.

[12] Воронцов К. В. Математические методы обучения по прецедентам (теория обучения машин). — URL: http://www.machinelearning.ru/wiki/images/6/6d/Voron-ML-1.pdf.

[13] Левитин А. В. Алгоритмы. Введение в разработку и анализ. Глава 3. Метод грубой силы: Поиск выпуклой оболочки. — М. : Вильямс, 2006. — С. 576.

[14] Мерков А.Б. Распознавание образов. Введение в методы статистического обучения. — М. : Едиториал УРСС, 2011. — С. 256.

[15] Шапиро Л., Стокман Д. Компьютерное зрение. — М. : Бином. Лаборатория знаний, 2006. — С. 752.

5. Приложение 1

В данном приложении приведены реализации методов локализации и классификации на языке Python, с использованием библиотек OpenCV, TensorFlow и Scikit-Learn.

```
import cv2
import numpy as np
from sklearn.cluster import AffinityPropagation


def cut_board(img):
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    canny = cv2.Canny(gray, 150, 255, 10)

    kernel = np.ones((3, 3), np.uint8)
    canny = cv2.dilate(canny, kernel, iterations=5)

    lines = cv2.HoughLines(canny, 1, np.pi / 2, 200)

    y_1 = 0
    y_2 = 10000
    for line in lines:
        for rho, theta in line:
            a = np.cos(theta)
            b = np.sin(theta)
            x0 = a * rho
            y0 = b * rho
            x1 = int(x0 + 1000 * (-b))
            y1 = int(y0 + 1000 * (a))
            x2 = int(x0 - 1000 * (-b))
            y2 = int(y0 - 1000 * (a))
            y_1 = max(y_1, y1)
            y_2 = min(y_2, y2)
    img = img[y_2+10:y_1-10, :]
    return img

def find_contours (img):

    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    histr = cv2.calcHist([gray], [0], None, [10], [0, 256])
    persum = 0
    s =  np.sum(histr)
    ind = 0
    for i, num in enumerate(histr):
        persum+=num/s
```

```
41          if persum > 0.045:
42              break
43          ind = i
44
45      ind += 1
46      kernel = np.ones((3, 3), np.uint8)
47      mask = cv2.inRange(gray, 0, 26*ind)
48      height, width, channels = img.shape
49      line_width = 10
50      line_color = (0, 0, 0)
51      cv2.line(mask, (0, 0), (width, 0), line_color, line_width)
52      cv2.line(mask, (0, height), (width, height), line_color, line_width)
53      cv2.line(mask, (0, 0), (0, height), line_color, line_width)
54      cv2.line(mask, (width, 0), (width, height), line_color, line_width)
55      dilate = cv2.dilate(mask, kernel, iterations=5)
56      er = dilate.copy()
57      contours = cv2.findContours(er, cv2.RETR_CCOMP, cv2.CHAIN_APPROX_SIMPLE)
58      con = contours[1]
59      cn = []
60      for cnt in (contours[2]):
61          for idx, hier in enumerate(cnt):
62              if hier[3] == -1:
63                  cn.append(con[idx])
64
65      cn1 = []
66      height, width, channels = img.shape
67      arp = width * height
68      for cnt in cn:
69          ar = cv2.contourArea(cnt)
70          per = cv2.arcLength(cnt, True)
71          if ar + per * 3 < arp and ar > 70:
72              cn1.append(cnt)
73              x = ar + per * 3
74      return cn1
75
76  def mech(img):
77      gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
78      kernel = np.ones((5, 5), np.uint8)
79      clahe = cv2.createCLAHE(clipLimit=100.0, tileGridSize=(15, 15))
80      gray1 = clahe.apply(gray)
81      sobel = cv2.Sobel(gray1, cv2.CV_64F, 0, 1, ksize=5)
82      gray1 = gray / gray1
83      gray1 = cv2.inRange(gray1, 0, 10)
84      gray1 = cv2.morphologyEx(gray1, cv2.MORPH_OPEN, cv2.getStructuringElement(
                                                cv2.MORPH_ELLIPSE, (5, 5)))
85      gray1 = cv2.morphologyEx(gray1, cv2.MORPH_OPEN, cv2.getStructuringElement(
```

```
                                                cv2.MORPH_ELLIPSE, (5, 5)))
    gray1 = cv2.morphologyEx(gray1, cv2.MORPH_OPEN, cv2.getStructuringElement(
                                                cv2.MORPH_ELLIPSE, (5, 5)))
    gray1 = cv2.pyrDown(gray1)
    gray1 = cv2.pyrDown(gray1)
    gray1 = cv2.pyrDown(gray1)
    gray1 = cv2.pyrUp(gray1)
    gray1 = cv2.pyrUp(gray1)
    gray1 = cv2.pyrUp(gray1)
    morphed = gray1
    gray1 = cv2.inRange(gray1, 0, 210)
    morphed = cv2.morphologyEx(morphed, cv2.MORPH_OPEN, cv2.
                                                getStructuringElement(cv2.
                                                MORPH_ELLIPSE, (5, 5)))
    gray1 = cv2.resize(gray1, (gray.shape[1], gray.shape[0]))

    params = cv2.SimpleBlobDetector_Params()
    params.minThreshold = 215  # the graylevel of images
    params.maxThreshold = 226
    detector = cv2.SimpleBlobDetector_create(params)
    keypoints = detector.detect(morphed)
    xy = []
    for kpt in keypoints:
        x, y = kpt.pt
        xy.append([int(x), int(y)])
    channel_count = img.shape[2]
    ignore_mask_color = (255,) * channel_count
    mask = np.zeros(img.shape, dtype=np.uint8)
    for p in xy:
        mask = cv2.circle(mask, (p[0], p[1]), 12, ignore_mask_color, -1, cv2.
                                                    LINE_AA)
    dil = cv2.morphologyEx(mask, cv2.MORPH_OPEN, cv2.getStructuringElement(cv2
                                                .MORPH_ELLIPSE, (5, 5)))
    dil = cv2.cvtColor(dil, cv2.COLOR_BGR2GRAY)

    contours = cv2.findContours(dil, cv2.RETR_CCOMP, cv2.CHAIN_APPROX_SIMPLE)
    con = contours[1]
    cn = []
    for cnt in (contours[2]):
        for idx, hier in enumerate(cnt):
            if hier[3] == -1:
                cn.append(con[idx])
    return cn


def cl_find(img):
```

```
126     xy = []
127     gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
128     sift = cv2.xfeatures2d.SURF_create()
129 
130     kernel = np.ones((3, 3), np.uint8)
131     img = cv2.dilate(img, kernel, iterations=3)
132     kp, dsc = sift.detectAndCompute(img, None)
133 
134     for kpt in kp:
135         x, y = kpt.pt
136         xy.append([x, y])
137     xy = np.asarray(xy)
138 
139     af = AffinityPropagation(preference=-50000).fit(xy)
140     cluster_centers_indices = af.cluster_centers_indices_
141     labels = af.labels_
142     n_clusters_ = len(cluster_centers_indices)
143 
144     restricted_labels = []
145     for k in range(n_clusters_):
146 
147         class_members = labels == k
148 
149         print(len(xy[class_members]))
150         if len(xy[class_members]) <= 10:
151             restricted_labels.append(k)
152     hulls = []
153 
154     for k in range(n_clusters_):
155 
156         if k in restricted_labels:
157             continue
158         class_members = labels == k
159 
160         hulls.append([cv2.convexHull(xy[class_members].astype(np.float32))])
161 
162     return hulls
```

Code Listing 1: Программная реализации методов локализации

```
import os, sys
import numpy as np
import tensorflow as tf
from alexnet import AlexNet
import cv2
from matplotlib import pyplot as plt
import matplotlib.patches as mpatches
import matplotlib.colors as colors
import localization as lc

config = tf.ConfigProto()
config.gpu_options.allow_growth = True

'''image'''
img_path = ''
'''path'''

'''images'''
img = cv2.imread(img_path)
img = lc.cut_board(img)
contours = lc.find_contours(img)
mechs = lc.mech(img)
vis = img.copy()
cn1 = contours
cn1.extend(mechs)
channel_count = img.shape[2]
ignore_mask_color = (255,) * channel_count
defects = []

learning_rate = 0.01
num_epochs = 10
batch_size = 128

dropout_rate = 0.5
num_classes = 11
train_layers = ['fc8', 'fc7', 'fc6']

display_step = 1

filewriter_path = "C:/TFLogs/tmp/finetune_alexnet/defects"
checkpoint_path = "C:/TFLogs/tmp/finetune_alexnet/"

if not os.path.isdir(checkpoint_path):
    os.mkdir(checkpoint_path)

x = tf.placeholder(tf.float32, [batch_size, 227, 227, 3])
```

```
47 y = tf.placeholder(tf.float32, [None, num_classes])
48 keep_prob = tf.placeholder(tf.float32)
49
50 model = AlexNet(x, keep_prob, num_classes, train_layers)
51
52 score = model.fc8
53
54 var_list = [v for v in tf.trainable_variables() if v.name.split('/')[0] in
                                          train_layers]
55
56 with tf.name_scope("cross_ent"):
57     loss = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(logits=score
                                              , labels=y))
58
59 with tf.name_scope("train"):
60
61     gradients = tf.gradients(loss, var_list)
62     gradients = list(zip(gradients, var_list))
63
64     optimizer = tf.train.GradientDescentOptimizer(learning_rate)
65     train_op = optimizer.apply_gradients(grads_and_vars=gradients)
66
67 for gradient, var in gradients:
68     tf.summary.histogram(var.name + '/gradient', gradient)
69
70 for var in var_list:
71     tf.summary.histogram(var.name, var)
72
73 tf.summary.scalar('cross_entropy', loss)
74
75 with tf.name_scope("accuracy"):
76     correct_pred = tf.equal(tf.argmax(score, 1), tf.argmax(y, 1))
77     accuracy = tf.reduce_mean(tf.cast(correct_pred, tf.float32))
78
79 tf.summary.scalar('accuracy', accuracy)
80
81 merged_summary = tf.summary.merge_all()
82
83 writer = tf.summary.FileWriter(filewriter_path)
84
85 saver = tf.train.Saver()
86
87
88 def get_img(img_path):
89
90     scale_size = (227, 227)
```

```
    images = np.ndarray([128, scale_size[0], scale_size[1], 3])
    img = cv2.imread(img_path)

    img = cv2.resize(img, (scale_size[0], scale_size[0]))
    img = img.astype(np.float32)

    images[0] = img
    return images

def get_images(defects):

    # Read images
    scale_size = (227, 227)
    images = np.ndarray([128, scale_size[0], scale_size[1], 3])
    for i, img in enumerate(defects):

        img = cv2.resize(img, (scale_size[0], scale_size[0]))
        img = img.astype(np.float32)

        images[i] = img
    return images

classes = []

with tf.Session(config=config) as sess:

    sess.run(tf.global_variables_initializer())
    writer.add_graph(sess.graph)
    model.load_initial_weights(sess)
    saver.restore(sess, 'C:/TFLogs/tmp/finetune_alexnet/model_epoch500.ckpt')

    for idx, cnt in enumerate(cn1):
        hull = cv2.convexHull(cnt)

        mask = np.zeros(img.shape, dtype=np.uint8)
        cv2.fillPoly(mask, [hull], ignore_mask_color)
        masked_image = cv2.bitwise_and(img, mask)
        defects.append(masked_image)

    images = get_images(defects)

    classification = sess.run(score, {x: images, keep_prob: 1.})

    res = tf.nn.l2_normalize(classification, 1, epsilon=1e-12, name=None)
    for idx, item in enumerate(defects):
        label = sess.run(tf.argmax(res[idx]))
```

```
137         preds = sess.run(res)[idx]
138 
139         keys = {0: 'mechanical defect', 1: 'rested knot', 2: 'corrupted knot',
140                 3: 'pin knot', 4: 'unknown defect', 5: 'massive knot',
141                 6: 'tar', 7: 'crack', 8: 'flawed knot', 9: 'fallen knot',
142                 10: 'darken'}
143         classes.append({'label': keys[label], 'color': label})
144 
145     patch = []
146     sortcols = []
147 
148     cols = {0: 'red', 1: 'green', 2: 'blue',
149             3: 'yellow', 4: 'teal', 5: 'cyan',
150             6: 'navy', 7: 'darkmagenta', 8: 'orange',
151             9: 'tan', 10: 'azure'}
152 
153     for idx, cnt in enumerate(cn1):
154         hull = cv2.convexHull(cnt)
155         name = classes[idx]['label']
156         col = cols[int(classes[idx]['color'])]
157 
158         color = colors.to_rgb(col)
159         color = [int(i*255) for i in color]
160         color.append(30)
161         color = tuple(color)
162         cv2.drawContours(vis, [hull], -1, color, 1, lineType=cv2.LINE_AA)
163         if col not in sortcols:
164             patch.append(mpatches.Patch(color=col, label=name))
165             sortcols.append(col)
166     patch = list(set(patch))
167 
168     plt.imshow(vis)
169 
170     plt.legend(bbox_to_anchor=(0., 1.02, 1., .102), loc=3, handles=patch,
171     ncol=10, mode=''expand'', borderaxespad=0.)
172     plt.xticks([]), plt.yticks([])
173 
174     plt.show()
175     sess.close()
```

Code Listing 2: Программная реализация классификации

```
import tensorflow as tf
import numpy as np

class AlexNet(object):

  def __init__(self, x, keep_prob, num_classes, skip_layer,
               weights_path = 'DEFAULT'):

    self.X = x
    self.NUM_CLASSES = num_classes
    self.KEEP_PROB = keep_prob
    self.SKIP_LAYER = skip_layer

    if weights_path == 'DEFAULT':
      self.WEIGHTS_PATH = 'bvlc_alexnet.npy'
    else:
      self.WEIGHTS_PATH = weights_path

    self.create()

  def create(self):

    conv1 = conv(self.X, 11, 11, 96, 4, 4, padding = 'VALID', name = 'conv1')
    pool1 = max_pool(conv1, 3, 3, 2, 2, padding = 'VALID', name = 'pool1')
    norm1 = lrn(pool1, 2, 2e-05, 0.75, name = 'norm1')

    conv2 = conv(norm1, 5, 5, 256, 1, 1, groups = 2, name = 'conv2')
    pool2 = max_pool(conv2, 3, 3, 2, 2, padding = 'VALID', name ='pool2')
    norm2 = lrn(pool2, 2, 2e-05, 0.75, name = 'norm2')

    conv3 = conv(norm2, 3, 3, 384, 1, 1, name = 'conv3')

    conv4 = conv(conv3, 3, 3, 384, 1, 1, groups = 2, name = 'conv4')

    conv5 = conv(conv4, 3, 3, 256, 1, 1, groups = 2, name = 'conv5')
    pool5 = max_pool(conv5, 3, 3, 2, 2, padding = 'VALID', name = 'pool5')

    flattened = tf.reshape(pool5, [-1, 6*6*256])
    fc6 = fc(flattened, 6*6*256, 4096, name='fc6')
    dropout6 = dropout(fc6, self.KEEP_PROB)

    fc7 = fc(dropout6, 4096, 4096, name = 'fc7')
    dropout7 = dropout(fc7, self.KEEP_PROB)

    self.fc8 = fc(dropout7, 4096, self.NUM_CLASSES, relu = True, name='fc8')

```

```
47
48
49    def load_initial_weights(self, session):
50
51      weights_dict = np.load(self.WEIGHTS_PATH, encoding = 'bytes').item()
52
53      for op_name in weights_dict:
54
55        if op_name not in self.SKIP_LAYER:
56
57          with tf.variable_scope(op_name, reuse = True):
58
59            for data in weights_dict[op_name]:
60
61              if len(data.shape) == 1:
62
63                var = tf.get_variable('biases', trainable = False)
64                session.run(var.assign(data))
65
66              else:
67
68                var = tf.get_variable('weights', trainable = False)
69                session.run(var.assign(data))
70
71
72  def conv(x, filter_height, filter_width, num_filters, stride_y, stride_x, name
                                                ,
73           padding='SAME', groups=1):
74
75    input_channels = int(x.get_shape()[-1])
76
77    convolve = lambda i, k: tf.nn.conv2d(i, k,
78                                         strides = [1, stride_y, stride_x, 1],
79                                         padding = padding)
80
81    with tf.variable_scope(name) as scope:
82
83      weights = tf.get_variable('weights',
84      shape = [filter_height, filter_width, input_channels/groups, num_filters])
85      biases = tf.get_variable('biases', shape = [num_filters])
86
87
88      if groups == 1:
89        conv = convolve(x, weights)
90
91      else:
```

```
92
93        input_groups = tf.split(axis = 3, num_or_size_splits=groups, value=x)
94        weight_groups = tf.split(axis = 3, num_or_size_splits=groups,
95        value=weights)
96        output_groups = [convolve(i, k) for i,k in zip(input_groups,
97        weight_groups)]
98
99        conv = tf.concat(axis = 3, values = output_groups)
100
101     bias = tf.reshape(tf.nn.bias_add(conv, biases), conv.get_shape().as_list()
                                                )
102
103     relu = tf.nn.relu(bias, name = scope.name)
104
105     return relu
106
107 def fc(x, num_in, num_out, name, relu = True):
108   with tf.variable_scope(name) as scope:
109
110     weights = tf.get_variable('weights', shape=[num_in, num_out], trainable=
                                                True)
111     biases = tf.get_variable('biases', [num_out], trainable=True)
112
113     act = tf.nn.xw_plus_b(x, weights, biases, name=scope.name)
114
115     if relu == True:
116       relu = tf.nn.relu(act)
117       return relu
118     else:
119       return act
120
121
122 def max_pool(x, filter_height, filter_width, stride_y, stride_x, name, padding
                                        ='SAME'):
123   return tf.nn.max_pool(x, ksize=[1, filter_height, filter_width, 1],
124                         strides = [1, stride_y, stride_x, 1],
125                         padding = padding, name = name)
126
127 def lrn(x, radius, alpha, beta, name, bias=1.0):
128   return tf.nn.local_response_normalization(x, depth_radius = radius,
129   alpha = alpha, beta = beta, bias = bias, name = name)
130
131 def dropout(x, keep_prob):
132   return tf.nn.dropout(x, keep_prob)
```

Code Listing 3: Построение сети AlexNet

```
 import os
import numpy as np
import tensorflow as tf
from datetime import datetime
from alexnet import AlexNet
from datagenerator import ImageDataGenerator

config = tf.ConfigProto()
config.gpu_options.allow_growth = True

train_file = 'wtrain.txt'
val_file = 'wtest.txt'

learning_rate = 0.01
num_epochs = 50
batch_size = 128

dropout_rate = 0.5
num_classes = 11
train_layers = ['fc8', 'fc7', 'fc6']

display_step = 1

filewriter_path = "C:/TFLogs/tmp/finetune_alexnet/defects"
checkpoint_path = "C:/TFLogs/tmp/finetune_alexnet/"

if not os.path.isdir(checkpoint_path):
    os.mkdir(checkpoint_path)

x = tf.placeholder(tf.float32, [batch_size, 227, 227, 3])
y = tf.placeholder(tf.float32, [None, num_classes])
keep_prob = tf.placeholder(tf.float32)

model = AlexNet(x, keep_prob, num_classes, train_layers)

score = model.fc8

var_list = [v for v in tf.trainable_variables() if v.name.split('/')[0] in
                                        train_layers]

with tf.name_scope("cross_ent"):
    loss = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(logits=score
                                            , labels=y))

with tf.name_scope("train"):
    gradients = tf.gradients(loss, var_list)
```

```
    gradients = list(zip(gradients, var_list))

    optimizer = tf.train.GradientDescentOptimizer(learning_rate)
    train_op = optimizer.apply_gradients(grads_and_vars=gradients)

for gradient, var in gradients:
    tf.summary.histogram(var.name + '/gradient', gradient)

for var in var_list:
    tf.summary.histogram(var.name, var)
tf.summary.scalar('cross_entropy', loss)

epoacc = []

with tf.name_scope("accuracy"):
    correct_pred = tf.equal(tf.argmax(score, 1), tf.argmax(y, 1))
    accuracy = tf.reduce_mean(tf.cast(correct_pred, tf.float32))

tf.summary.scalar('accuracy', accuracy)

merged_summary = tf.summary.merge_all()

writer = tf.summary.FileWriter(filewriter_path)

saver = tf.train.Saver()

train_generator = ImageDataGenerator(train_file,
                                     horizontal_flip=True, shuffle=True)
val_generator = ImageDataGenerator(val_file, shuffle=False)

train_batches_per_epoch = np.floor(train_generator.data_size / batch_size).
                                       astype(np.int16)
val_batches_per_epoch = np.floor(val_generator.data_size / batch_size).astype(
                                       np.int16)

with tf.Session(config=config) as sess:

    sess.run(tf.global_variables_initializer())

    writer.add_graph(sess.graph)

    model.load_initial_weights(sess)

    print("{} Start training...".format(datetime.now()))
    print("{} Open Tensorboard at --logdir {}".format(datetime.now(),
                                                      filewriter_path))
```

```
89
90      for epoch in range(num_epochs):
91
92          print('{} Epoch number: {}'.format(datetime.now(), epoch + 1))
93
94          step = 1
95
96          while step < train_batches_per_epoch:
97
98              batch_xs, batch_ys = train_generator.next_batch(batch_size)
99
100             sess.run(train_op, feed_dict={x: batch_xs,
101                                           y: batch_ys,
102                                           keep_prob: dropout_rate})
103
104             if step % display_step == 0:
105                 s = sess.run(merged_summary, feed_dict={x: batch_xs,
106                                                         y: batch_ys,
107                                                         keep_prob: 1.})
108                 writer.add_summary(s, epoch * train_batches_per_epoch + step)
109
110             step += 1
111
112         print('{} Start validation'.format(datetime.now()))
113         test_acc = 0.
114         test_count = 0
115         for _ in range(val_batches_per_epoch):
116             batch_tx, batch_ty = val_generator.next_batch(batch_size)
117             acc = sess.run(accuracy, feed_dict={x: batch_tx,
118                                                 y: batch_ty,
119                                                 keep_prob: 1.})
120             test_acc += acc
121             test_count += 1
122         test_acc /= test_count
123         print('{} Validation Accuracy = {:.4f}'.format(datetime.now(),
                                                        test_acc))
124
125         epoacc.append((epoch, test_acc))
126         print(epoacc[epoch])
127
128         val_generator.reset_pointer()
129         train_generator.reset_pointer()
130
131         print('{} Saving checkpoint of model...'.format(datetime.now()))
132
```

```
133         checkpoint_name = os.path.join(checkpoint_path, 'model_epoch' + str(
                                                    epoch + 1) + '.ckpt')
134         save_path = saver.save(sess, checkpoint_name)
135 
136         print("{} Model checkpoint saved at {}".format(datetime.now(),
                                                    checkpoint_name))
```

Code Listing 4: Fine-Tuning сети

```
1  import numpy as np
2  import cv2
3  
4  class ImageDataGenerator:
5      def __init__(self, class_list, horizontal_flip=False, shuffle=False,
6                   mean = np.array([104., 117., 124.]), scale_size=(227, 227),
7                   nb_classes = 11):
8  
9          self.horizontal_flip = horizontal_flip
10         self.n_classes = nb_classes
11         self.shuffle = shuffle
12         self.mean = mean
13         self.scale_size = scale_size
14         self.pointer = 0
15 
16         self.read_class_list(class_list)
17 
18         if self.shuffle:
19             self.shuffle_data()
20 
21     def read_class_list(self,class_list):
22         with open(class_list) as f:
23             lines = f.readlines()
24             self.images = []
25             self.labels = []
26             for l in lines:
27                 items = l.split()
28                 self.images.append(items[0])
29                 self.labels.append(int(items[1]))
30 
31             self.data_size = len(self.labels)
32 
33     def shuffle_data(self):
34 
35         images = self.images.copy()
36         labels = self.labels.copy()
37         self.images = []
```

```
38          self.labels = []
39
40          idx = np.random.permutation(len(labels))
41          for i in idx:
42              self.images.append(images[i])
43              self.labels.append(labels[i])
44
45      def reset_pointer(self):
46
47          self.pointer = 0
48
49          if self.shuffle:
50              self.shuffle_data()
51
52      def next_batch(self, batch_size):
53
54          paths = self.images[self.pointer:self.pointer + batch_size]
55          labels = self.labels[self.pointer:self.pointer + batch_size]
56
57          self.pointer += batch_size
58
59          images = np.ndarray([batch_size, self.scale_size[0], self.scale_size[1
                                                    ], 3])
60          for i in range(len(paths)):
61              img = cv2.imread(paths[i])
62
63              if self.horizontal_flip and np.random.random() < 0.5:
64                  img = cv2.flip(img, 1)
65
66              img = cv2.resize(img, (self.scale_size[0], self.scale_size[0]))
67              img = img.astype(np.float32)
68
69              img -= self.mean
70
71              images[i] = img
72
73          one_hot_labels = np.zeros((batch_size, self.n_classes))
74
75          for i in range(len(labels)):
76              one_hot_labels[i][labels[i]] = 1
77
78          return images, one_hot_labels
79
80      def get_img(self, img_path):
81
82          images = np.ndarray([1, self.scale_size[0], self.scale_size[1], 3])
```

```
83          img = cv2.imread(img_path)
84
85          img = cv2.resize(img, (self.scale_size[0], self.scale_size[0]))
86          img = img.astype(np.float32)
87
88          images[0] = img
89          return images
```

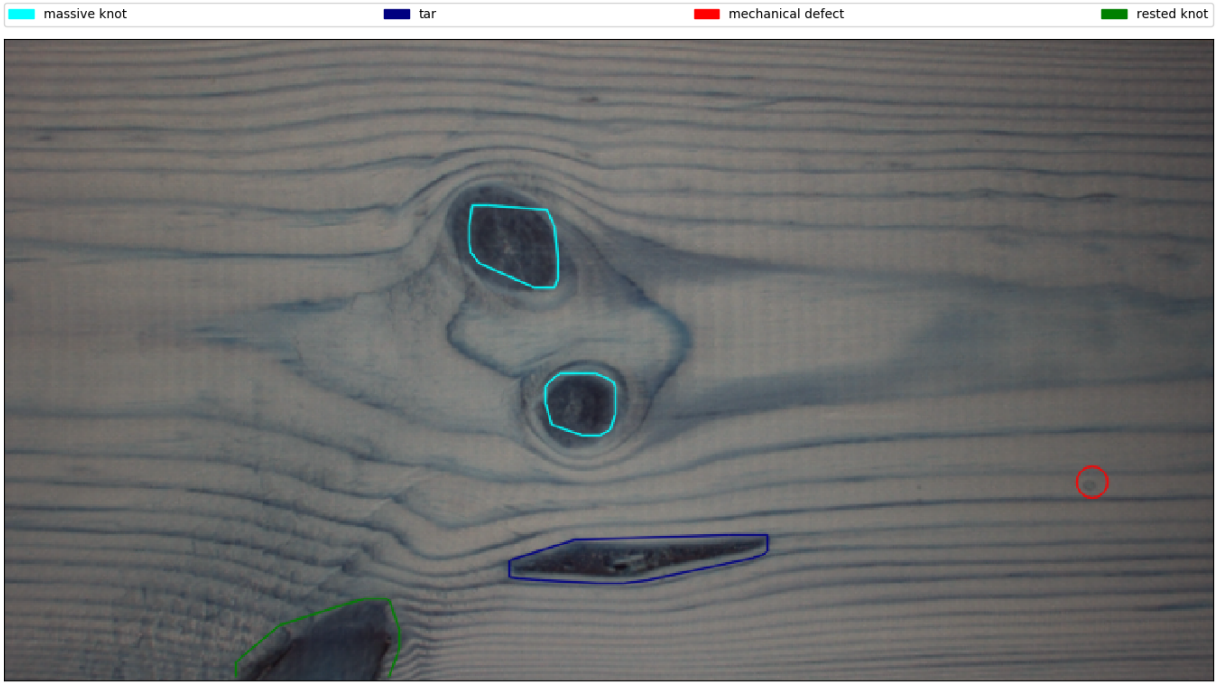
Code Listing 5: Генератор пакетов для обучения классификатора

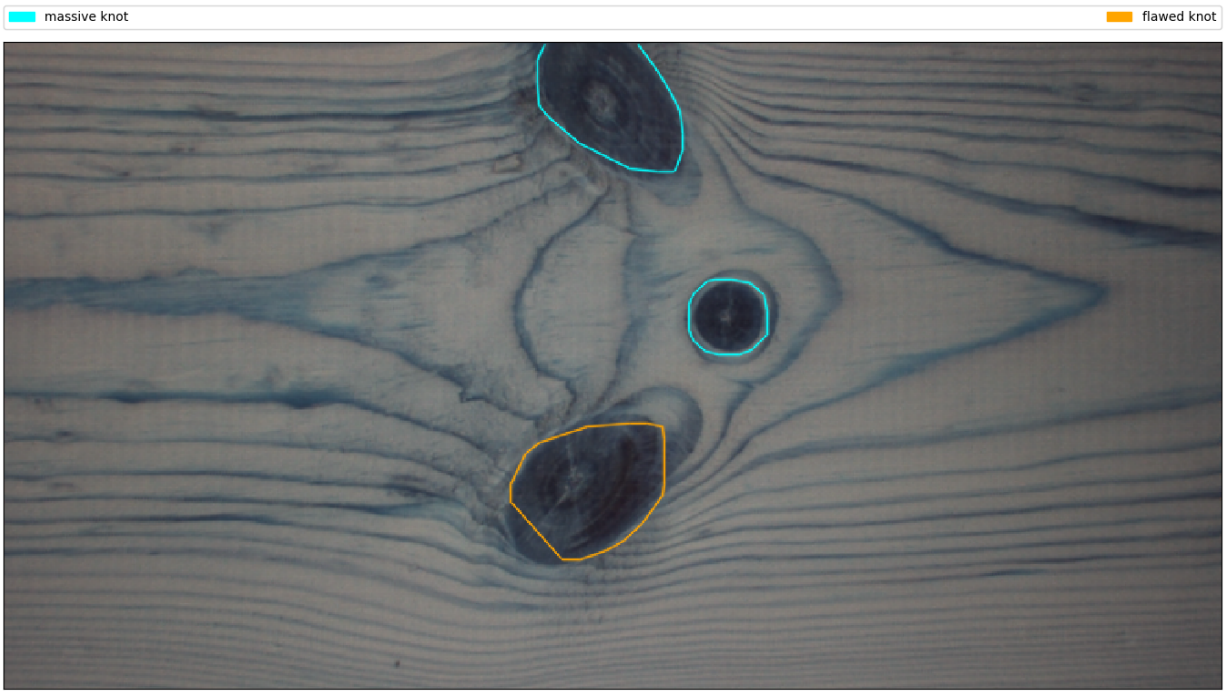
6. Приложение 2

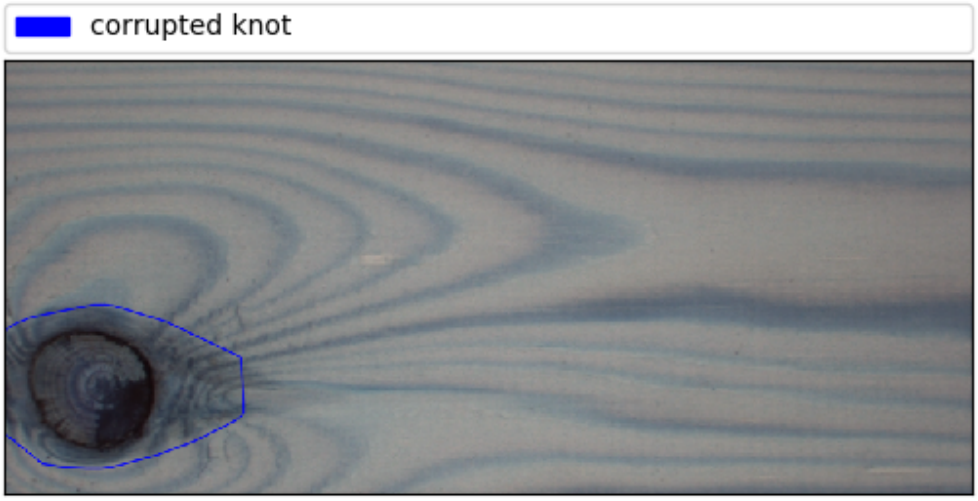
В данном приложении содержится таблица определяемых дефектов и больше скриншотов результатов созданной программы.

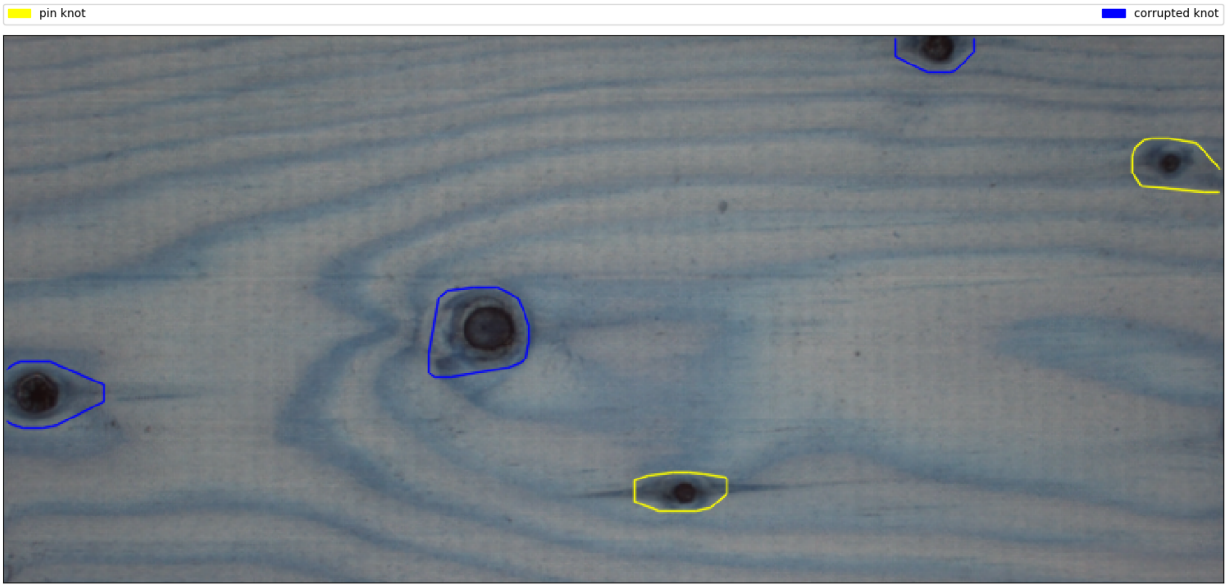
Таблица 1: Классификация дефектов

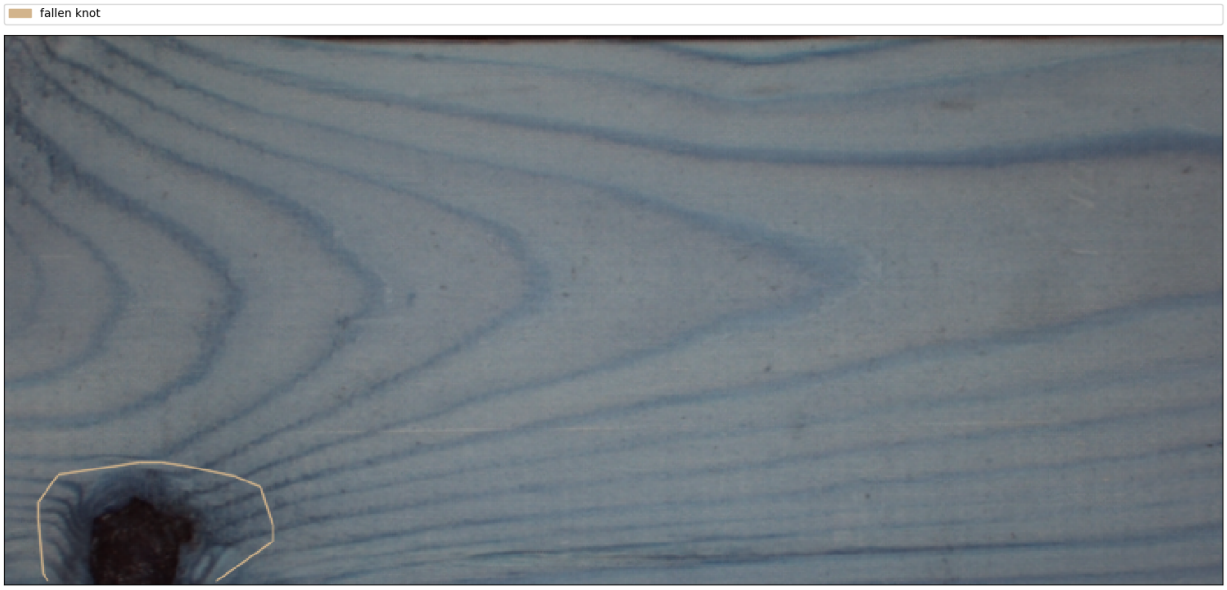
Номер дефекта	Класс дефекта	Расшифровка
0	mechanical defect	Механическое повреждение
1	rested knot	Несросшийся сучок
2	corrupted knot	Вросший сучок
3	pin knot	Крошечный сучок
4	unknown defect	Неизвестный дефект
5	massive knot	Здоровый сучок
6	tar	Смоляной карман
7	crack	Трещина
8	flawed knot	Сучок с дефектом
9	fallen knot	Выпавший сучок
10	darken	Потемнение




massive knot
flawed knot


corrupted knot


pin knot
corrupted knot


fallen knot